软 件 工 程 系 列

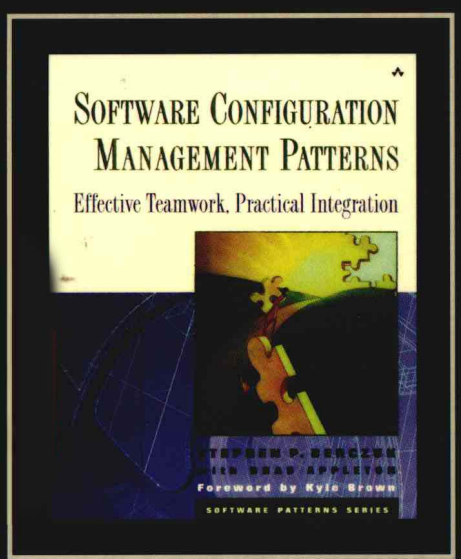
PEARSON
Addison Wesley

Software Configuration Management Patterns

Effective Teamwork, Practical Integration

软件配置管理模式

[美] Stephen P. Berczuk 著
Brad Appleton
Kyle Brown 序
黄明成 译



- Amazon 五星图书，清晰讲解配置管理软件
- 透彻说明中小型开发团队中的 SCM 应用
- 针对实际软件开发，强调相关重点和难点
- 讨论常见 SCM 工具并解释实现本书模式过程

Pearson Education
中国电力出版社

软 件 工 程 系 列

Software Configuration Management Patterns

Effective Teamwork, Practical Integration

软件配置管理模式

[美] Stephen P. Berczuk 著
Brad Appleton
Kyle Brown 序
黄明成 译

中国电力出版社
www.infopower.com.cn

Software Configuration Management Patterns: Effective Teamwork, Practical Integration (ISBN 0-201-74117-2)
Stephen P. Berczuk, Brad Appleton

北京市版权局著作权合同登记号 图字：01-2003-3828

图书在版编目（CIP）数据
软件配置管理模式 /（美）勃克扎等编；黄明成译. 北京：中国电力出版社，2004
（软件工程系列）
ISBN 7-5083-2189-8
Ⅰ. 软... Ⅱ. ①勃...②黄... Ⅲ. 软件－配置－管理 Ⅳ. TP31
中国版本图书馆 CIP 数据核字（2004）第 024604 号

丛 书 名：软件工程系列
书　　名：软件配置管理模式
编　　著：（美）Berczuk, Appleton
翻　　译：黄明成
责任编辑：姚贵胜
出版发行：中国电力出版社
　　地址：北京市三里河路6号　　邮政编码：100044
　　电话：（010）88515918　　传　　真：（010）88518169
印　　刷：北京丰源印刷厂
开　　本：787×1092　1/16　　印　　张：11　　字　　数：236千字
书　　号：ISBN 7-5083-2189-8
版　　次：2004 年 7 月北京第 1 版　　2004 年 7 月第 1 次印刷
定　　价：22.00 元

译者的话

感谢汪亚文先生，在本书出版的第一时间，在本书的书名还没有在 Amazon 网站露面的时候，就把它介绍到中国来，使中国读者能在第一时间一睹《软件配置管理模式》的风采。

这本书来得很及时。

随着经济迅速发展，随着信息化步伐加快，计算机应用越来越广、越来越深，应用项目越来越多、越来越大、越来越复杂、越来越灵活、越来越交叉。奋战在信息化第一线的开发人员和管理人员迫切期望跟上新的变化。

本书能把你一步带到前沿。

请注意，本书不是“软件配置管理”，而是“软件配置管理模式”。

其实，本书不仅是“软件模式”，而且是真正的“软件模式语言”。

这很可能是介绍到我国的第一部完整的软件模式语言著作。

关于模式和模式语言，你大概早已耳熟能详。不过，即使第一次接触，也不必担心。本书第 3 章有一段简短而精彩的介绍，从 Alexander 的《Pattern Language》到 Gamma 等 4 位作者的《Design Patterns：Elements of Reuseable Object-Oriented Software》，到本书的模式语言，一看就知道了。

从软件工程到软件模式，从软件模式到软件模式语言，是信息时代人类认识的又一尝试、又一探索。

Hunt Andrew 和 David Thomas 在《The Pragmatic Programmer：From Journeyman to Master》中写道：

人们常把软件开发比作建筑工程。用建筑作比喻蕴涵着下面几个步骤：

（1）建筑师绘出蓝图。

（2）承建方挖掘基础，建设地上结构，布线布管并进行最后装修。

（3）房客高高兴兴地搬进去住。如果发现问题，就叫维修人员来修。

但是，软件完全不是这样。软件不像建筑，而更像园艺——比混凝土有活力。你按照最初的计划和条件，在花园里种了很多花木。有些茁壮成长，有些注定变为堆肥。你可以变更苗木的相对位置，以充分利用光和影、风和雨的交相辉映。蔟叶过密的枝条可以分栽或修剪；不协调的花色可以移到更具审美情趣的地方。你拔除杂草，给需要特殊

照顾的植株施肥。你不断地关注花园的生机与健康，并根据需要，对土壤、植物品种和布局进行调整。

Hunt Andrew 和 David Thomas 又写到：

从结构化程序设计，历经主程序员项目组、CASE 工具、瀑布模型、螺旋模型、Jackson、实体关系图、Booch Clouds、OMT、对象库和 Cood/Yourdon，到今天的 UML，计算机应用从来不缺少打算使编程更像工程的方法。每一种方法都聚集一批追随者，而且每一种方法都会流行一段时间。接着，每一种又被下一种取代。在所有这些方法中，也许只有第一种——结构化程序设计——存在的时间最长。

有些开发者，在淹没了许多项目的大海里漂流，依然紧抱着最新的时尚，就像失事船只的受害者紧抓着漂来的木块。每当有新东西漂过时，他们都费力地游过去，希望比现在的更好。可是，不管漂过的东西从各方面看有多么好，开发者仍在无目的地漂流。

动态的世界，动态的需求，动态的开发，动态的管理。信息化已经取得的辉煌成就和信息化更加壮丽的诱人前景都在催人奋进，都在促使人们改变思想方式，不再是苦苦追寻一种公认的、放之四海而皆准的最佳软件开发方法论，而是脚踏实地、深入细致地分析与研究软件开发活动和与软件开发活动有关的其他活动的种种模式以及这些模式的相互联系，以便针对具体的项目，根据具体的环境、条件、资源、要求，动态地选择、动态地调整、动态地创造在此时此地最适宜的最佳方案，流畅地谱写具体项目成功与胜利的篇章，并为今后更多的成功与胜利打好可持续发展的基础，这也许就是软件模式与模式语言的真谛。

Robert L. Glass 在《Facts and Fallacies of Software Engineering》中列举了关于软件工程的 55 个“事实”和 5 加 5 个“谬见”。赫然列在第一条的是：

软件工作中最重要的因素是程序员的质量。

第二条则更加语出惊人：

最好的程序员比最差的程序员最多强 28 倍。

可见，所有的最佳实践都应该以人为本。让最好的程序员充分发挥他 28 倍的强势与潜力，让最差的程序员也能在流畅的环境超常发挥，实现 5 倍、10 倍或 28 倍的飞跃。

这也许就是本书的副题“有效的合作、实用的集成”所蕴涵的精髓。

感谢中国电力出版社，使我有机会担任本书的翻译。本书内容新颖，很多名词术语尚未见到经过国家权威审定或民间约定俗成的规范泽法。为了避免混淆，下面列出几个最没有把握的术语：

active codeline *活动码线*

active development 活动开发
agile software development 敏捷软件开发
branch 支线
branching 分支
build 构造
code base 码基
codeline 码线
codeline policy 码线策略
codeline retire 码线退休
configuration status accounting 配置状态会计
configuration audit 配置审计
glacial development 冰河开发
include/exclude approach 包含/排除法
mainline 主线
maintenance release 维护版
merge ancestry 归并世系
named stable base 命名稳定基
peopleware 人件
precheck-in 预检入
real version 实版
refactoring 重构
release engineering 发布工程
release line 版本线
release-prep code line 发布准备线
rework 重做
survival rule 生活经验
tip of codeline 码线末端
transaction-oriented 面向事务
virtual version 虚版

其中不妥或书中误译、漏译之处，请各位老师和朋友们指正。

谢谢!

黄明成

2004 年 5 月 1 日

序

熟悉我的工作的读者或许会想，为什么搞 J2EE 软件体系结构的专家竟会为论述软件配置管理（Software Configuration Management，SCM）的书作序。毕竟，这两门学科不能再分开了，难道不是吗？J2EE 体系结构似乎高高在上，而 SCM 也许被视为在软件开发中地位低下。实际上，没有什么比这更背离事实真相了。多年来，我常常发现，那些在 J2EE 应用体系结构上遇到问题的顾客，通常在 SCM 上也遇到了严重的问题。

这种离奇的巧合，有两层原因。首先，许多人通常很难迅速适应变化——例如放弃一套不再适用于像 J2EE 这样的新环境的体系结构实践，或者放弃在一种环境行之有效，但未必在所有的环境都行之有效的软件开发过程。这样，他们就会以为，如果他们的 SCM 过程对前一个项目行之有效，就一定对当前的项目也行之有效——而不顾设计与构造这两个项目所使用的技术、时间框架（timescale）与方法也许完全不同的事实。

其次，人们往往想靠一小组简单的规则支配他们的全部行动。然而，采取过于简单的方法通常会在抽象与现实交会的地方遇到问题。无论问题是理解为什么特定的 J2EE 构造（例如 Entity EJB）在一种情况下行之有效，而在另一种情况下却不行；还是理解为什么让开发者有自己的，能在其中进行开发与集成的私用工作区是重要的（毕竟，你迟早得把他们的代码加以集成），问题都是一样的。在这两种情况下，简单的规则（使用 Entity Bean；使用构造脚本）的确是好建议，但它必须在经验的熔炉中经受锻炼，因为在未经锻炼之前，它太脆弱，无法应用。

通过最近 20 年关于混沌和复杂性理论的研究，数学家与科学家们开始发现，虽然根据太少的和过于简单的规则构造的系统通常迟钝而单调，但只要增加很少几个规则，便常常可以得到惊人的复杂与美妙的系统。这些系统即使受到外力的严重扰乱，仍能自行重组，使总体构架保持完整。你手里的这本书就提供一组具有这种柔韧性的 SCM 规则。

Steve 和 Brad 提出了把 SCM 作为模式系统对待的成熟建议。正如他们早些时候有力地揭示的，模式系统的实力不在于各个模式本身，而在于模式之间的关系网。作者开发出模式的连锁网络，覆盖了最常见的 SCM 实践。然而，更重要的是，他们说明，SCM 面临的问题不是任何一个模式可以独自完全解决的——你需要仔细地考虑各个 SCM 实践与其他实践的联系，以免作茧自缚。

例如，你也许想提前看一下他们在第一个模式——“主线”（第 4 章）中给出的绝妙

建议。这个貌似平凡的建议（开发者应在单一、稳定的码基上工作）正是我发现被许多组织——包括那些在实现过程中已经花费了数百万美元的大型的、成功的公司——在某种程度上忽略了的东西。这是常识，非常实用的常识，而这正是它的难得之处。

同样，在“私用工作区”（第 6 章）和“私用系统构造”（第 8 章）中给出的建议，简直和使得现代的 Java 集成开发环境（例如 VisualAge for Java 和 IBM 的 WebSphere Studio）如此有用和如此流行的两个关键思想一模一样。当有人问我（差不多每天有人问），为什么开发者应当选择这样的集成开发环境，而不是用传统的代码编辑程序和编译程序在命令行进行开发时，这些工具不仅允许而且积极鼓励这种开发风格的事实，是使我能够用简单的措词表达我的建议的关键因素。

所以，我相信，你会像我一样发现本书有用，有启发。自从几年前，这些模式首次在程序模式语言（Pattern Languages of Programs，PLoP）会议上发表以来，我已经向人们介绍过本书的一些模式，而且我发现，它们对构筑坦诚的、有建设性的、关于如何以正确的方式实施 SCM 的论坛是无价之宝。这些模式已经在解决需要靠技巧与智谋来慎重处理的对顾客的承诺时，成为我劈开复杂的 SCM 问题的戈尔迪结的利剑——我希望，你也能很快开始挥舞这把利剑。

——Kyle Brown

《Enterprise Java Programming with IBM WebSphere》的作者

前 言

软件配置管理不是我的工作。我不是软件配置管理人员。我不是搞组织工作的人。然而，我早些时候发现，作为软件开发人员，理解组织、软件体系结构和软件配置管理对做好我的工作是必不可少的。我还发现，从这个角度看待软件工程很有意义。我构造软件系统，而配置管理是构造软件系统时非常重要却常被忽视的组成部分。在本书中，我希望能说明如何避免我遇到过的一些问题，使你和你的项目组能更有效地构造系统。

我大概应该解释一下，我所谓的软件配置管理（SCM）人员与构造软件系统的人员是什么意思。按照公式化形象，配置管理人员和工具与控制有关。他们保守，喜欢缓慢的、可预见的进展。与组织中占“多数”的开发者相比，他们是“少数”。软件工程师（同样按照公式化形象）则不拘小节。他们要赶快构造东西，并且自信在任何情况下都能按他们的方式编码。这些都是极端公式化的形象，而且，根据我的体验，优秀的软件工程师和优秀的版本/质量保证/配置管理人员有共同的目标：致力于交付优质系统，并尽量避免浪费精力。

良好的配置管理实践不是按时构造系统的银弹，也不是模式、极限编程（Extreme Programming，XP）、统一过程（Unified Process）或你也许听说过的其他任何东西。然而，它是被大多数人由于害怕“过程”——往往是由于过去的痛苦经验（Weigers 2002）——而忽略的工具箱的组成部分。

本书描述一些常见的软件配置管理实践。本书对那些在小项目组工作，觉得不能尽其所能有效地使用软件配置管理的软件开发者特别有意义。我们描述的技法（technique）与具体的工具无关。正像采用任何一组模式或最佳实践一样，能否自如地运用这些模式也许与你所使用的工具是否明确地支持它们有关。

为什么我写这本书

我的软件开发生涯是在波士顿地区的一个研发小组开始的。在工作中，除了遇到许多有趣的技术问题之外，还有个新的进展：与母公司所在地（纽约州 Rochester 市）的小组搞联合开发项目。这种体验使我刚参加工作就认识到，软件开发不仅涉及良好的设计

与良好的编码实践，而且涉及同一小组，甚至不同城市的不同小组的成员之间的协调。我们组牵头设置如何共享代码及开发过程的其他人工制品的机制。我们使用通常的东西使合作比较轻松，例如开会、远程会议和电子邮件发送清单等。我们设置本地小组（和远程小组）共享代码的软件配置管理系统的方式，在使协作比较轻松方面起了很大的作用。

为波士顿小组设置 SCM 过程的人，使用似乎一直在他们的职业生涯中屡试不爽的技法。当我转到其他组织时，我惊异地发现，有那么多地方还在为同一些常见的问题——我们早就知道有很好的解决方法的问题——而拼搏。因为我过去一直在几家刚创办不久的，我刚去时只有一两年历史的公司工作，所以情况更是如此。在新创办的公司，一两年往往是正在招募足够的人员，却很难达到协调与共识的阶段。

工作几年后，我发现了模式。当时，Erich Gamma、Richard Helm、Ralph Johnson 和 John Vlissides 刚完成了他们的著作《Design Patterns》（Gamma 等 1995），而 Hillside 集团正在筹备第一届 PLoP 会议。模式的想法有很大的力量，因为它致力于在正确的时间，用正确的方法解决问题；而且，因为模式是跨学科的：它们不是只关注具体领域或具体语言的编码技法，而是从各个角度（从编码到项目组）看待如何构造软件的问题。我在 PLoP 会议的各种研讨会上发表了许多论文，论述在设计、编码与配置管理交会之处的模式（Berczuk 1995，1996a，1996b；Appleton 等 1998；Cabrera 等 1999；Berczuk 和 Appleton 2000）。

在一次 PLoP 会议上，我遇到了 Brad Appleton，他是比我更强的 SCM 专家。我们合写了一篇关于分支模式的论文（Appleton 等 1998），论述了 SCM 的一个方面。在同行们大力鼓励之下，我们开始写作本书。

我希望本书能帮助你避免一些常见的错误。或者让你知道有这些途径；或者给你提供文档，使你能用来把你已经知道的途径向你们组织内的其他人进行解释。

谁应该阅读本书

我希望每个构造软件和使用配置管理系统的人都能从本书获益。配置管理问题的细节因你所构造的系统类型、项目组人员多少和你的工作环境而异。因为不大可能写出针对每个人的需要和使每个人感兴趣的书，所以我不得不仔细地选择我的话题。本书对那些在没有很多已定义过程的中小型组织中构造软件或管理软件项目的人最有价值。如果你在小公司、新创办的公司或大型组织中的小项目组工作，你就会从本书受益最多。即使你们组织已经有明确定义的、似乎妨碍工作进展的繁复的过程，你也可以用本书的模式更好地致力于某些关键的 SCM 任务。

如何阅读本书

“引言”部分介绍软件配置管理的一些基本概念和软件配置管理图使用的记法。第一部分提供与 SCM 和模式有关的背景信息。第 1 章介绍本书所有的软件配置管理概念。第 2 章论述影响你决定使用哪一类 SCM 环境的一些因素。第 3 章介绍模式的概念、本书的模式及它们之间的联系。第二部分介绍各种模式，说明 SCM 面临的问题和解决常见的 SCM 问题的方法。第 1 章和第 2 章还定义了本书针对的总的问题。要了解模式如何互相配合，应阅读第 3 章，以便对模式语言有个总的认识。

读过前三章以后，就可以浏览第二部分的模式，可以先阅读你认为有兴趣的模式，然后再阅读与你的问题有关的模式。也可以依次阅读各个模式，搞清它们之间的互相联系。

每章的导言和章末的“未解决的问题”节可能包含对本书出现的其他模式的参考，用“活动开发线（第 5 章）”这样的形式表示。

由于涉及领域很广，所以有些上下文和“未解决的问题”不包含对本书或其他地方的其他模式的参考，原因是在编写本书时尚未在文献上见到它们。在这种情况下，你会看到可能涉及什么模式的说明。

材料来源

本书的许多材料来源于我、Brad Appleton、Ralph Cabrera 和 Robert Orenstein 为历届 PLoP 会议撰写的论文。这些模式已作过很大的修改，但还是应该提到这些论文，以感谢他们为这一成果所做的贡献：《Streamed Lines：Branching Patterns for Parallel Software Development》（Appleton 等 1998）、《Software Reconstruction：Patterns for Reproducing the Build》（Cabrera 等 1999）、《Configuration Management Patterns》（Berczuk 1996b）。

关于照片

每章开头的照片均来自美国国会图书馆。所有的照片都是在 20 世纪上半叶拍摄的。只有两幅照片（第 5 章“活动开发线”和第 8 章“私用系统构造”的照片）例外，它们分别来自《大萧条时代至第二次世界大战》和《由大萧条至第二次世界大战的美国》这

两组收藏品。选择这些照片的原因，是想给模式提供直观的类比。软件是抽象的概念，但我们解决的许多问题，特别是有关项目组的问题，是与现实世界的问题类似的。而且，我对照片和历史一直有浓厚的兴趣。

——Steve Berczuk，
Arlington，Massachusetts，June 2002
steve@berczuk.com
http://www.berczuk.com

合作者前言

为什么我和 Steve 合写这本书

1987 年，为了付最后一年的大学学费，我作为兼职软件工具开发者，开始了软件开发生涯。不知道什么缘故，从此结下“不解之缘”，因为自那时起，我一直在做某种形式的工具开发（尤其是 SCM 工具），即使有时它不是我的主要工作。我甚至为商品 SCM 工具供应商工作过（时间不长），当时，我的部分工作是保持在竞争中的领先地位。所以，我尽可能多地搜罗市场上其他 SCM 工具的有关知识。即使更换工作之后，我还在继续追踪 SCM，并经常在 Internet 上参加各种工具的兴趣小组。

有一度，我渴望提高 SCM 环境的最高技术水平，赶上所有的最新研究成果。很快，就因为“最高技术水平”与“最高实践水平”之间的巨大鸿沟而灰心。我认定，最好是帮助提高实践水平，使现有的工具得到更好的利用。此后不久，我发现了软件模式和模式社区。显然，这些人对广为传播的、反复出现的软件设计最佳实践中，哪些是中肯的分析、哪些是无稽之谈已经有了重大的发现。

当时，在设计模式社区中，几乎没有任何人尝试编写 SCM 模式。毕竟，对许多程序员来说，如果把软件开发比作“建筑工程”，则 SCM 只是其中的“管道敷设”：每个人都承认需要它，但谁也不想不由自主地涉足过深，以免难以自拔。他们只要它起作用，而不愿被迫受它更多的打扰。

没有过多长时间，我就和 Steve Berczuk 联络上了。我们（还有 Ralph Cabrera）一起写了几篇关于 SCM 模式的论文，作为我的 ACME 项目的一部分（http://acme.bradapp.net/），后来又决定写这本书。我们希望这本虽然很薄，但紧紧围绕有关集成与合作的核心模式的书，能帮助心诚的开发者兼项目领导，在成功地领导与协调项目组的通力合作及把工作成果集成为优质软件的过程中节节胜利、不断进步。

感谢我妻子 Maria 的无限深情与支持（及我们的女儿 Kaeley）。感谢我父母的鼓励。同时感谢我以前的经理 Arbela 的鼓励、支持与友谊。

——Brad Appleton
Arlington Heights，Illinois，June 2002
brad@bradapp.net
http://www.bradapp.net

致 谢

如果你曾经读过书中的致谢，就会知道出版一本书需要的人远不仅仅是作者。虽然我过去也知道这一点，但通过本书（我的第一本书）的出版，使我认识得更加清楚了。我要感谢编辑 Debbie Lafferty 在指导我们编写我们的第一本书时的耐心与热情。同时感谢文稿编辑 Deborah English，她的明察秋毫与娴熟技巧使我能以比我所能想到的更好的方式表达本书的思想。我还要感谢 Addison-Wesley 公司的另一些人，本书得以出版，是和他们在本书的文字写好以后投入的大量工作分不开的：生产编辑 Amy Fleischer；封面设计 Karin Hansen；版式设计 Dede Cummings，她非常耐心地让我们看她的各种设计；排版员 Reuben Kantor；校对员 Barbara Ames；索引作者 Nancy Fulton 和负责营销工作的 Chris Guzikowski 与 Kate Saliba。

我还要感谢对手稿提出意见与建议的每一个人，包括：Heini Avela、Hisham Alzanoon、Bruce Angstadt、Stanley Barton、David Bellagio、Phil Brooks、Kyle Brown、Frank Buschmann、Thomas Dave、Bernard Farrell、Linda Fernandez、Jeff Fischer、William Hasling、Kirk Knoernschild、Dmitri Lapyguine、McDonald Michael、James Noble、Damon Poole、Linda Rising、Alan Shalloway、Eric Shaver、Michael Sheetz、Dave Spring、Marianne Tromp、Bob Ventimiglia、Ross Wetmore 和 Farrero Xavier Verges。

最后，我必须感谢 Gillian Kahn，她是我在许多事情上的合作伙伴，她的意见与建议、见识与耐心对我完成这个项目都是非常宝贵的。

引 言

这里描述本书使用的一些基本概念、记法和术语。软件配置管理（SCM）词汇在不同的上下文中以不同的方式使用，这里的定义不是这些术语的使用方式的全面论述。只要可能，我们尽量使用常见的术语。同时，我们还简要地介绍版本控制实践，并推荐一些进一步的阅读材料。

关键概念和术语

“软件配置管理”包括配置标识、配置控制、状态记录、复审、构造管理、过程管理和合作等要素（Dart 1992）。组织所采用的 SCM 实践在总体上定义组织如何构造与发布产品及如何标识与跟踪变更。本书关心 SCM 中对编写代码和实现代码的特征与变更的人的日常工作有直接冲击的内容。

开发者遇到的概念（不一定是这个名称）有“工作区”（workspace）、“码线”（codeline）和“集成”（integration）。

“工作区”是开发者保存为了完成任务而需要的所有人工制品的地方。具体地说，工作区可以是开发者工作场所的磁盘上的目录树，也可以是通过工具在抽象的空间维护的一组文件。在正常情况下，工作区与这些人工制品的特定版本相联系。工作区还应该有能够用它的内容来构造可执行的人工制品的机制。例如，如果你在用 Java 开发，则工作区应该包括：

- 以适当的包结构编排的源码（.java 文件）
- 测试用的源码
- Java 库文件（.jar 文件）
- 不用你构造的本机接口库文件（例如，Windows 中的.dll 文件）
- 定义如何把.java 文件构造为可执行文件的脚本

有时，工作区是在集成开发环境（IDE）的上下文中管理的。工作区还和一条或多条码线相联系。

“码线”是这组源文件与组成某个软件组件的其他人工制品随着时间而变更的进展过程。每当你在版本控制系统中变更文件或其他人工制品时，都在创建该人工制品的“修

订版”（revision）。码线包含沿着一条路径发展的各个人工制品的每个版本。

在任一时间点，码线的快照（snapshot）包含该码线上各个组件的各个修订版。图 I-1 表示，在一个时间点，你有第一版的 file1.java 和 file2.java。在下一次码线有变更时，得到的码线“末端”（tip）状态包含 file1.java 的第一修订版和 file2.java 的第二修订版[1]。包含码线上每个组件的修订版集合的码线的任一快照，是码线的“配置”（configuration）。[2] 任一赋予特殊名字或编号的配置是码线的“版本”（version）。如果你决定把一个版本标识为特殊版本，就定义一个“标号”（label）。例如，可以给进入一个版次的修定版集合加标号。

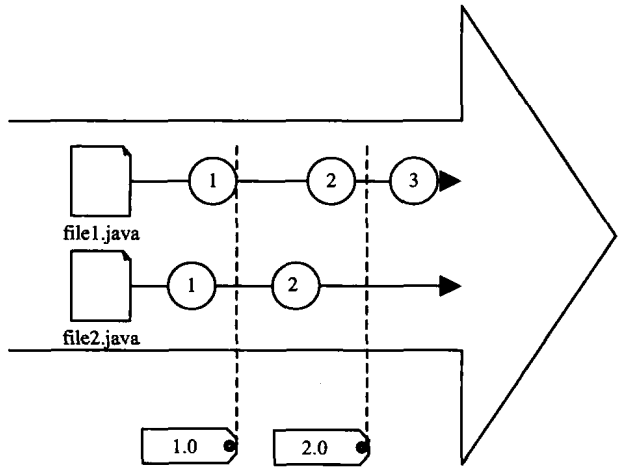


图 I-1　码线及其组件

在最简单的情况下，你也许只有一条码线，其中包括你的所有的产品代码。码线上的组件按它们自己的速率发展，并且有我们可以标识的修订版。你可以通过标号标识码线的版本。码线的版本是快照，其中包括到达标号那一时间点时组件的所有的修订版。

一件产品可以有不止一条码线，但每条码线只包含一组连贯的工作。各码线可以有不同的用途，而且可以用各种码线的快照的可识别配置填充你的工作区。例如，你可以把第三方代码放入一条码线，在另一条码线上进行活动开发，把作为内部产品对待的内部工具放入第三条码线，如图 I-2 所示。每条码线还有与之相联系的“策略”（policy）。这些策略定义了该码线的用途及你可以在何时与如何进行变更的规则。

随着码线的发展，你也许发现有些工作是从码线的意图派生出来的。在这种情况下，

1　原文如此。——译者

2　通常，你也可以通过“标记”（tag）组件的不同修订版来标识码线的版本——例如，第一版的 file2.java 和第三版的 file1.java。但是有另外像这样更直观的标识配置的方法。

你可能要把文件分支，使它可以独立于原来的开发自行发展。文件的“分支”（branch）是文件的修订版，它以主干版本为起点，独立地发展。图 I-3 表示，在第二修订版之后，有人创建分支，并把文件变更为修订版 2.1、2.2……表示分支的常用记法是在“.”后加个次版本号，表示分支出来的修订版是以主干上的主修订版为基础的。创建分支的理由可能是，例如，你要开始在产品的新版本上工作，但又要能解决已发布的版本中的问题。在这种情况下，你可以创建分支，表示已发布的版本，同时在主干上继续你的工作。你在分支上进行的某些变更，可能还需要回到主干，所以你进行“合并”（merge），把变更从分支集成到主干上来。图 I-3 用从修订版 2.2 到修订版 3 的虚线表示这一过程。

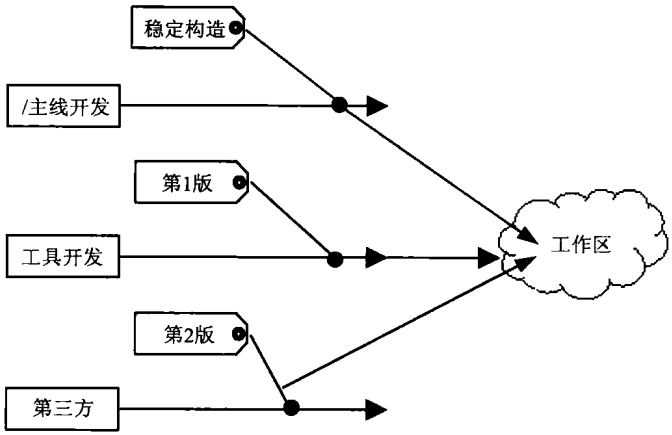


图 I-2　用不同码线填充工作区

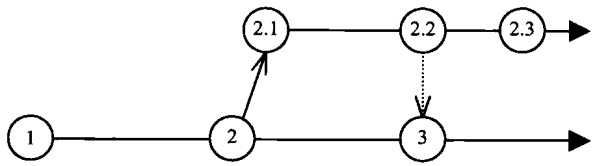


图 I-3　分支单个文件及与干线合并

合并可借助于能识别上下文文本区别的工具在一定程度上自动地进行，但你往往需要正确地理解把变更加以合并的意图。

你往往需要分支整条码线中的一组互有联系的文件，而不是单个文件。在这种情况下，版本是指把整条码线作为一个单位的版本，其中码线的版本包括码线中所有的修订版在那一时间点的版本，如图 I-4 所示。

每当你变更码线中的任何东西时，都创建了该码线末端的一个新配置。码线的这个新快照在概念上可能隐含着有新版本需要标识。实际上，码基的大多数用户不需要用惟一的